



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/776,909	02/11/2004	Louis R. Degenaro	YOR919990064US2 (8728-258)	3057
46069	7590	09/21/2007	EXAMINER	
F. CHAU & ASSOCIATES, LLC 130 WOODBURY ROAD WOODBURY, NY 11797			CHOI, WOO H	
			ART UNIT	PAPER NUMBER
			2189	
			MAIL DATE	DELIVERY MODE
			09/21/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

**BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES**

Application Number: 10/776,909
Filing Date: February 11, 2004
Appellant(s): DEGENARO ET AL.

MAILED

SEP 21 2007

Technology Center 2100

Frank V. DeRosa
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed May 18, 2007 appealing from the Office action mailed October 18 2006.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is generally correct. However, because of Appellants' apparent misunderstanding (as evidenced by arguments regarding claims 42 and 46) of the relationship between a probability of a statement execution and a likelihood of a value of a cacheable entity changing, which is directly related to the desirability of performing at least one cache transaction, the Examiner will attempt to provide additional explanations regarding the claim limitations "probability that at least one statement

will execute”, “desirability of performing at least one cache transaction”, and “likelihood of a value of a cacheable entity changing” to clarify the relationship among them for easier understanding of the claims and teachings of references as related to the claimed limitations.

First of all, the Examiner notes that Appellants use the terms “probability” to represent “likelihood” (see specification, page 36, lines 10 – 11, “a probability is determined which represents the likelihood that the detected statement will be executed”). The term “probability” seems to be used as a synonym for “likelihood”.

“**A probability that at least one statement will execute**” depends on the structure of the program being analyzed. As Appellants explained (Appeal Brief, page 3, last sentence), a statement that is outside a conditional branch will execute unconditionally with probability of 1 (certainty). On the other hand, a program statement within a conditional branch may or may not execute depending on which of the two possible branches is taken by the execution of the program. The probability that a statement in one branch will execute is p , where p is the probability that the program execution will take this branch. The probability that a statement in the other branch will execute is $1-p$. The Examiner notes that Appellants carefully avoided the use of the word “calculating” and instead consistently used a broader term “determining” in the specification and the claims. Therefore, Appellants do not disclose, nor do they claim, calculation of a probability value. The disclosed and claimed determination seems to be an assessment of whether it is likely to take one branch or the other branch.

“**A desirability of performing at least one cache transaction**” depends on whether the value of a cacheable data object will change and whether the data object is likely to be accessed in the near future. A cache is a high-speed memory (faster than the main memory) with

Art Unit: 2189

relatively small capacity (because it is more expensive than slower main memory) used in a computer system to temporarily keep data items likely to be accessed again to improve memory access speed. In a typical caching system, when a data item is to be accessed the system searches the cache memory. If the data item needed is in the cache memory, it is retrieved from the faster cache (this is called a cache “hit” in the art of caching). If not, the system must retrieve the data from the slower main memory (a cache “miss”). Keeping only data items likely to be accessed again improves the overall system performance. Therefore, it is not desirable to keep data items used only once or infrequently accessed in cache because they occupy limited space that should be populated with more frequently accessed data. It is also not desirable to keep obsolete data or a data item that is about to be replaced with a new value. Caching systems use various known techniques, such as least recently used, least frequently used, and other cache replacement algorithms to attempt to replace infrequently accessed data with fresh data. These systems also use cache invalidation techniques to flag obsolete data for eviction or “flushing” to make room for more desirable data. Some systems try to anticipate the likelihood of needing a data item, by analyzing access patterns or program structures, and populate the cache (i.e., make caching decisions) prior to the actual need for the data.

As Appellants explained (Appeal Brief, page 4), if a data item is likely to change its value 1) it is not desirable to populate this data item in cache if it is not in cache already, and 2) it may be desirable to invalidate if it is occupying space in cache because it is likely to become obsolete. Thus, one factor that affects the desirability of a caching transaction is whether the **value of a data object is likely to change**, which depends on the actual statement (other factors include whether a data object is likely to be needed soon or whether the object is likely to be infrequently

accessed). For example, in a statement $X = Y + Z$, the value of X is likely to change, if executed, depending on the values of Y and Z. On the other hand, values of Y and Z will not change by the execution of this statement. Whether the value of X may change depends in turn on whether the statement will execute at all. If the statement is not executed, the value of X will not change. Therefore, the likelihood of the value of X changing depends on both the statement (structure of the statement) itself and whether the statement will execute (structure of the program, i.e., probability that the program will take the branch that contains the statement).

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

5,940,857	Nakanishi et al.	8-1999
5,774,685	Dubey	6-1998
6,073,129	Levene et al.	6-2000

Cytron et al. "Automatic Management of Programmable Caches" Proceedings of the 1988 International Conference on Parallel Processing, 1998, pp 75-84

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

1. Claims 1 – 3, 5, 8 – 11, 14, 18 – 20, 22, 25 – 28, 42 and 46 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. The language of the claim raises a question as to whether the claims is directed merely to an abstract idea that is not tied to a technological art, environment or machine which would result in a practical application producing a concrete, useful, and tangible result to form the basis of statutory subject matter under 35 U.S.C. 101. Even when automated, the claims seem to be directed to a method of planning/preparing with no practical application of the plan. This still amounts to a machined manipulated abstract idea, which is considered non-statutory.
2. Claims 1 – 48 are rejected under the judicially created doctrine of double patenting over claims 1 - 37 of U. S. Patent No. 6,725,333 since the claims, if allowed, would improperly extend the "right to exclude" already granted in the patent. Because the Appellant concedes the propriety of the double patenting rejection, the statement of the rejection is not repeated here.
3. Claims 42 – 44, 46 – 48 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the

Art Unit: 2189

relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention.

With respect to claims 42 and 46, according to the specification the probability of a value of a cacheable entity changing is based on the probability that a statement will execute, not the other way around as claimed (see specification page 36, lines 14 – 23, the probability of a value of a cacheable entity changing depends on the probability that the statement containing the cacheable entity will execute because if the statement is not executed the value cannot change; a change in value can only be caused by the execution of the statement, and whether the statement will execute depends on the structure of the program, i.e., whether the program will take the branch containing the statement, and not on whether the statement will change some value when executed).

At page 5, lines 10 – 14, the specification states “a probability is determined which represents the likelihood that the detected statements will be executed (i.e., the likelihood that one or more cachable entities will change due to execution of the statement) (step 601).” When read in isolation, this statement seems to suggest that the probability of execution is the same as the likelihood that a cachable entity will change, but when read in conjunction with the preceding sentence (page 5, lines 5 – 10) and figure 6, it is fairly clear that the specification is consistent with the Examiner’s understanding that the likelihood that one or more cachable entities will change depends on both the probability that the statement containing the cachable entity will execute (program structure, spec. page 5, lines 10 - 20) and that the statement is structured to modify the value of the cachable entity (statement structure, spec., lines 5 – 10). See section (5) Summary of Claimed Subject Matter above..

With respect to claims 43, 44, 47 and 48, caching, updating and invalidating are to be performed depending on the likelihood of a value of a cachable entity changing exceeding or not exceeding a threshold, as opposed to the probability of a statement execution exceeding or not exceeding a threshold (see specification page 37, lines 1 – 15).

4. Claims 1, 5, 10, 18, 22, 27, 35, 39, 40, 41 and 45 are rejected under 35 U.S.C. 102(b) as being anticipated by Nakanishi *et al.* (US Patent No. 5,940,857, hereinafter “Nakanishi”).

Nakanishi discloses an automated method for managing a plurality of cachable entities, comprising the steps of:

analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed (abstract, col. 4, lines 22 – 44, an instruction is analyzed to determine whether it is desirable to cache the instructions in the next block, i.e. if the statement in this block is executed, it determines the likelihood of needing the next block for continued execution and it is desirable to cache a block that is needed for execution because accessing data from cache instead of the main memory improves the overall system performance by allowing faster data/instruction access);

augmenting the program code with additional code to assist in determining the desirability of performing the at least one cache transaction (col. 27, lines 7 – 14, branch indication bit is added to the branch instruction to read ahead, or pre-fetch into cache, i.e., cache transaction, instructions in the predicted branch path of the program being analyzed; as

mentioned above, it is desirable to have instructions in the branch path likely to be taken in cache before they are actual needed for execution),

determining a probability that the at least one statement will execute (col. 27, lines 7 – 14, branch destination block with high possibility of being read out from the main memory, i.e., high probability of execution, is determined and indicated with a branch indication bit added to the branch instruction; see also col. 26, lines 54 – 64);

determining the desirability of performing the at least one cache transaction based on a probability that the at least one statement will execute (see above, see also col. 6, lines 43 – 50; if is probable that the program will take a branch path it is desirable to have the instruction and data in that branch path because they are likely to be needed in the near future).

5. Claims 1, 2, 4 – 6, 8, 18, 19, 21, 23, 25, 35, 36, 38 – 41, and 45 are rejected under 35 U.S.C. 102(b) as being anticipated by Dubey (US Patent No. 5,774,685).

Dubey discloses a system for managing a plurality of cachable entities, comprising:

a program analyzer to analyze program code and determine if there is at least one statement which may affect a desirability of performing at least one cache transaction, if the at least one statement is executed (col. 3, lines 58 – col. 4, line 4, a program is analyzed to identify points in the program where cache misses are likely to occur, it is desirable to avoid cache misses by prefetching data objects likely to cause cache misses);

the program analyzer determining a probability that the at least one statement will execute (abstract, speculative prefetches associated with conditional branches by its very nature is a determination that it is probable that a particular branch will be taken) and determining the

Art Unit: 2189

desirability of performing the at least one cache transaction based on a probability that the at least one statement will execute (col. 4, lines 31 – 44, the compiler determines the desirability of prefetching data based on a speculation that the prefetched data will actually be needed, i.e. based on the probability or likelihood that the branch that requires the data will be taken, and inserts a STOUCH instruction at a prefetch point) determines; and

a cache manager for performing the at least one cache transaction if it is determined to be desirable (col. 2, lines 21 – 27, instructions identified by the STOUCH instruction, i.e., those determined to be desirable, are fetched in the instruction cache).

6. Claims 1 – 8, 10, 14 – 25, 27, 31 – 48 are rejected under 35 U.S.C. 102(b) as being anticipated by Cytron *et al.* (Automatic Management of Programmable Caches, Proceedings of the 1988 International Conference on Parallel Processing, 1988, pp. 75 – 84).

Cytron discloses a system for managing a plurality of cachable entities, comprising:

a program analyzer (page 229, right column, second paragraph) to analyze program code and determine if there is at least one statement which may affect a desirability of performing at least one cache transaction, if the at least one statement is executed (page 231, 2.0 Algorithms, the algorithm analyzes a program to determine cacheability of variables, see page 230, left hand column, 1.1 Software-Controlled Caches for definition of cacheability or degree of desirability of caching, see also page 232, 2.2 Posting Values to Global Memory, Cyclon discloses a program code sequence that should cause, i.e. highly desirable, a posting of a cached value, i.e. cache transaction);

Art Unit: 2189

the program analyzer determining a probability that the at least one statement will execute (1.2 Execution model, the program analyzer of Cytron's disclosure analyzes DO loops to achieve parallelism, statements in a DO loop are always executed, i.e., probability of execution is determined to be 1, see also 2.1, Processor-Crossing dependencies, the analyzer also determines a probability that a statement will execute in different processors) and the desirability of performing the at least one cache transaction based on a probability that the at least one statement will execute (figure 4, the probability of executing a Write(P_i, X) -> Read(P_j, X) pattern in a DO loop shown in figure 4 is 1, and the analyzer determines the desirability by augmenting the code with a POST operation code); and

a cache manager for performing the at least one cache transaction if it is determined to be desirable (this is accomplished when the augmented program in figure 4 is executed).

7. Claims 9 and 26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Cytron in view of Levine *et al.* (US Patent No. 6,073,129).

Cytron discloses all of the limitations of the parent claims as discussed above. However, Cytron does not specifically disclose the use of SQL code and that the code includes a SET statement. On the other hand Levine specifically discloses RDMS system that uses SQL codes including a SET statement (col. 29, line 57).

It would have been obvious to one of ordinary skill in the art, having the teachings of Cytron and Levine before him at the time the invention was made, to use the RDMS teachings of

Art Unit: 2189

the computer system with caching of Levine in the computer system with caching of Cytron, in order to in apply Cytron's automatic management of programmable caching method in a practical and widely used real life application such as the one disclosed by Levine.

(10) Response to Argument

Rejections under 35 U.S.C. 101

Claim 1 is directed to a method for managing a plurality of cacheable entities. Yet the claimed steps of this “method of managing” cacheable entities only comprise analyzing a program code, determining a probability and a desirability. Not even a single step directed to the actual management of cacheable entities is claimed by Appellant. The claim does not even require any cacheable entity that the method is supposed to manage. In the absence of any step that actually uses the results of analysis and determinations for managing cacheable entities, for example by making caching decisions based on the results, analysis of code and determination of probabilities amount to automation of algorithmic steps that do not impart any useful practical and concrete results by themselves. Algorithms that are not put to any practical use are not statutory even when they are automated. A method for managing cacheable entities that does not recite any step to actually manage cacheable entities and does not even require any cacheable entity that the method is supposed to manage cannot produce any useful, tangible, and practical result of managing cacheable entities.

Claim 18 is directed to a program storage device tangibly embodying an executable program to perform method steps recited in claim 1. Claim 18 is non-statutory for the same reason as claim 1.

Double Patenting Rejections

Appellant concedes the propriety of double patenting rejections.

Rejections under 35 U.S.C. 112, 1st paragraph**Claims 42 and 46**

Claims 42 and 46 recite the limitation “determining said probability based on a likelihood of a value of a cacheable entity changing.” “[S]aid probability” refers to the “probability that the at least one statement will execute.” Appellant contends that this limitation is supported by passage in the specification at page 36, lines 17-23. However, the passage actually supports the rejection. According to the passage, if the probability that a statement $x = a * b$ will execute is high, the compiler would conclude that x has a high probability of changing. In other words, the likelihood that the value of x (a cacheable entity) will change depends on the probability of the statement executing, not the other way around as claimed. The value of x cannot change if the statement is not executed. The value of x has no effect on whether the statement containing the variable x will execute as claimed, because the execution of the statement containing the variable x depends on the structure of the program and other variables (in the cited passage above, the value of y) that affect branching decisions. On the other hand, the probability that the statement containing the variable x will execute will always affect the value of x , because the value of x will be recalculated every time the statement is executed. Appellant is confused about the causal relationship between the probability that a statement containing a variable will execute and the likelihood that the value of the variable in the statement will change.

Claims 43, 44, 47, and 48

These claims depend from claims 42 and 46. Therefore, they inherited the logical flaw of their parents discussed above. In addition, the claims require caching, invalidating, or updating a

value of said cacheable entity, if said probability is greater than or equal to a threshold.

Appellant seems to be conflating the probability of a statement executing and a likelihood of a cacheable entity changing, which are two separate values. The claims require making certain caching decisions based on the result of comparing the probability of a statement executing with a threshold. This is not supported by the specification. At page 37, lines 1-15, the specification clearly discloses that caching decisions are made based on the probability or likelihood of the value of the cacheable entity changing. The caching decisions are not made based on the result of comparing the probability that a statement will execute with a threshold as claimed.

Rejections under 35 U.S.C. 102

Nakanishi Reference

Appellant asserts that Nakanishi does not disclose “analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the least one statement is executed.” At col. 4, lines 22 – 44, Nakanishi clearly discloses this claimed limitation. Nakanishi discloses that instructions in a program block is analyzed to predict if it will be necessary to read in the next block from the main memory, and if so to cache the next block to reduce the probability of cache miss (Nakanishi calls this cache “error”). Appellant contends that this finding is fundamentally flawed for various reasons without articulating those reasons by clearly discussing why Nakinishi’s teachings do not read on the language of the claim. Appellant’s only real argument seems to be that the purpose of Nakanishi’s analysis of program code is not the same as Appellant’s, which is irrelevant in anticipation analysis.

Appellant does not dispute that Nakanishi discloses program code analysis. There is no dispute that this analysis of the program code is performed to decide whether another block is to be speculatively read into a cache prior to its actual need. Appellant admits that this decision is based on determination of presence of a branch instruction and its predicted outcome (Appeal Brief, page 17, Nakanishi col. 18, lines 56 – 65). Appellant also admits that Nakanishi teaches that “[t]he instruction analysis section (5) analyzes the content of a read block of instruction to determine if there is a branch instruction or not, and if there is a branch instruction, and branch destination address is calculated (see e.g., Col. 19, lines 12-20)” (Appeal Brief, page 17, 1st full paragraph, last sentence). In one embodiment of the Nakanishi’s invention, the decision to speculatively cache the next block (“at least one cache transaction”) depends on the presence of a branch instruction (“if there is at least one statement”) and whether resulting destination address is outside the region (“desirability of performing at least one cache transaction”, not desirable to cache the next block if it is outside the region) of the current and the next block (see col. 2, lines 40 – 46). Nakanishi clearly discloses all of the elements of the limitation.

Appellant further argues that “the Examiner fails to address the specific claim language that the program code is analyzed to determine if there is at least one statement which can affect the desirability of performing at least one cache transaction, if the at least one statement is executed” (Appeal Brief, page 16, last paragraph – page 17, first paragraph). Appellant argues that “if the detected statement is not executed, it does not affect a desirability of performing a transaction” and that “[t]he Examiner does not explain how Nakanishi teaches this aspect of the claimed invention.” This is a frivolous argument. Appellant is arguing limitations that are not in the claims. The claims do not specify what happens when the detected statement is not executed.

Art Unit: 2189

While claims are interpreted in light of the specification, limitations are not imported from the specification.

Appellant also seems to be arguing that Nakanishi does not teach the claimed step of “determining a probability that the at least one statement will execute” without articulating in a reasonable and a convincing manner as to why this is so. Nakanishi discloses this throughout the specification, but claim 8 (col. 30 lines 41 – 52) summarizes this aspect of the disclosure. The claim states in part “branch predict means for predicting whether the branch instruction is more likely than not to execute a branch operation.” In other words, the branch prediction analysis circuit determines whether the probability of executing a particular branch operation is greater than 0.5 (50 %). Thus the analysis means determines the probability that the instructions at the destination of the branch operation will execute. The same analysis also determines the complementary probability that the instructions following the branch statement (i.e., instructions to be executed if decision is made not to perform a branch operation) will or will not execute.

Dubey Reference

Prefetching is a techniques used to improve memory access performance (see col. 1, lines 14 – 21). In a prefetch, a data item or an instruction is fetched into the cache (which is a faster access memory than the main memory) for possible future use prior to its actual need. This practice improves memory access speed by supplying the data needed from the cache and avoids processor wait time associated with slow main memory access cycles (because the needed data item is already in the faster cache memory). Dubey teaches a method of speculatively prefetching objects into a cache and making other caching decisions, such as discarding or

retaining prefetched cached data, based upon evaluation of speculative conditions (see title of the invention and abstract).

Appellant's arguments seem to depend entirely on the anticipating Dubey reference not using the same words used by Appellant in the claims rather than the substance of the disclosure. For example, Appellant admits that Dubey teaches that compile time analysis is performed to determine locations within a program wherein cache misses are likely to occur at run-time (Appeal Brief, page 22). Yet, Appellant argues that Dubey does not teach "determining the desirability of performing at least one cache transaction based on the probability that the at least one statement will execute." A cache miss occurs when an instruction or data object needed for execution is not present in the cache memory and needs to be fetched from the slower main memory. When the compiler analyzes a program and identifies a location within the program where a cache miss is likely to occur, it is because the compiler has determined that it is likely that an instruction requiring a cached object (data or instruction) will execute and that the needed object will not be present in the cache. The compiler inserts STOUCH prefetch instruction at an identified prefetch point because it was deemed desirable for the identified object to be cached.

Appellant argues that "with [Dubey's] process, when a given statement is detected, there is no determination as to the probability that the statement will execute. In contrast it is assumed that the statement will execute (i.e., speculative prefetch) and the STOUCH instruction for the given statement is generated to initiate a speculative prefetch and specify compile time conditions that are evaluated against the run-time outcomes to determine whether to discard or maintain the prefetch instruction/data" (Appeal Brief, page 22). However, Appellant fails provide any explanation or evidence to support the assertion that Dubey's method simply

Art Unit: 2189

assumes that the statement will execute. Appellant admits that Dubey teaches a compiler that identifies locations where cache misses are **likely** to occur (Appeal Brief, page 20, second paragraph). The word “likely” denotes probabilities, not assumptions. Appellant also admits that Dubey teaches **speculative** prefetch. A speculation is not an assumption. Dubey’s speculation is based on the likelihood of a cache miss. As explained above cache misses can only occur if a statement requiring data object or instruction to be retrieved is executed when the required item is not present in cache. Therefore, the likelihood of a cache miss is directly related to the probability that the statement needing access to an item not in cache will execute. Appellant’s discussion of STOUCH instruction itself relates to Dubey’s improvements over conventional speculative prefetches, and not to the assessment or determination of the likelihood of a cache miss occurring.

Cytron Reference

In spite of explanations contained in the statement of rejections of how Cytron’s disclosure anticipates “analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed” limitation, Appellant merely offers repetitive allegation that the rejection does not explain how Cytron’s disclosure is even remotely related to the limitation. First of all, the Examiner notes that Cytron’s disclosure of the algorithm that determines the cacheability of variables (page 231, 2.0 Algorithms) is enough to meet this limitation. Cytron discloses a three-step algorithm that identifies all variables and marks them as “cacheable”, “temporarily cacheable”, or “non-cacheable”. These steps identify the desirability of caching each of the

Art Unit: 2189

variables if the statement containing the variable is executed. A variable marked as “cacheable” is highly desirable to cache (see the definition of “cacheable” at page 230, left column). A statement containing a “cacheable” object or variable affects the desirability of conducting a caching transaction, because it contains an object that can always be cached. A statement containing a “non-cacheable” object also affects the desirability of performing a cache transaction because such a variable should never be cached.

Cytron discloses “determining a probability that the at least one statement will execute”, because Cytron’s execution model mainly concerns the analysis of Do loops for parallelizing operations in a parallel computing environment (page 230, 1.2 Execution Model). In a Do loop, every statement is executed. Therefore, each time a Do loop is encountered, the probability of the statements in the loop being executed is 1 and the compiler analyzes the code to insert caching operation instructions based on this certainty (or probability of that is determine to be 1).

The limitation “determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement will execute” is also met because when a variable referred to in a statement in a Do loop is analyzed, it is categorized into one of the three possible cacheability categories (or degrees of desirability).

Cytron, also discloses the claim limitation in another way. Cytron’s analyzer analyzes program code (see for example, page 233, Figure 8; see also Figures 3 and 5) and augments the code by inserting cache transaction operations such as POST (posting of a cache value to a global memory) and INVALID (invalidation of a cache entry). The analyzer insert these cache management instructions because it has determined that it is desirable to conduct these cache transactions based on the statements in the Do loop for which the probability of execution is 1.

Rejections under 35 U.S.C. 103

While Appellant has argued 35 U.S.C 103 rejections of claims 9 and 26 under a separate heading, there is no new argument. Appellant's arguments are based on the allegation that Cytron does not anticipate the parent claims. Therefore, claims 9 and 26 stand or fall with their parent claims.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

Art Unit: 2189

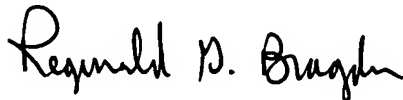
For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

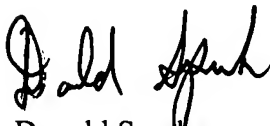


Woo H. Choi
Primary Examiner
GAU 2189

Conferees:



Reginald Bragdon
Supervisory Patent Examiner
GAU 2189



Donald Sparks
Supervisory Patent Examiner
GAU 2187